Planning & Scheduling Assignment 3 Report

Patrik Kupco

June 6, 2025

Name:	Patrik Kupco
Registration Number:	ies25210
Email:	kupco.patrik.16@gmail.com

1 Exercise 1: Facility Location Problem

1.1 Problem Description

The facility location problem involves placing a specified number of facilities on a grid to minimize the maximum distance from any client to their nearest facility, while respecting distance constraints between facilities and clients.

Key constraints:

- Facilities must be placed on a `grid_size` × `grid_size` grid
- Minimum distance required between any two facilities (`distance_between_facilities`)
- Each facility has a minimum distance requirement from all clients (`min_distance`)
- Objective: minimize the maximum distance from any client to their nearest facility
- Distance calculated using Manhattan distance

1.2 Solution Code

```
array[CLIENTS] of var DIST_RANGE: client_min_distance;
var DIST_RANGE: max_of_mins;

array[FACILITIES, COORDS] of var POSITIONS: facility_loc;
```

At the beginning of the code there are two decision variables to solve this problem. First one is the `client_min_distance`, which holds all of the minimum distances for each client to their nearest facility, and `max_of_mins`, which holds the maximum of the minimum values, which will be the variable for which the problem will be optimized.

1. Minimum Distance Between Facilities Constraint

```
constraint forall(i, j in FACILITIES where i < j) (
  abs(facility_loc[i, POSX] - facility_loc[j, POSX]) +
  abs(facility_loc[i, POSY] - facility_loc[j, POSY]) >
  distance_between_facilities
);
```

Here is a constraint that describes the minimum Manhattan distance between each facility location.

2. Minimum Distance Between Facilities/Clients Constraint

```
constraint forall(f in FACILITIES, c in CLIENTS) (
  abs(facility_loc[f, POSX] - client_loc[c, POSX]) +
  abs(facility_loc[f, POSY] - client_loc[c, POSY]) > min_distance[f]
);
```

Here is a constraint that describes the minimum Manhattan distance between each combination of facility and client location.

3. Minimum Facility/Client Distance Calculation

```
constraint forall(c in CLIENTS) (
  client_min_distance[c] =
    min([abs(facility_loc[f, POSX] - client_loc[c, POSX]) +
         abs(facility_loc[f, POSY] - client_loc[c, POSY]) | f in FACILITIES])
);
```

This constraint is responsible for modelling the minimum possible distance between each client/facility location. This is where all of the relevant calculations happen.

4. Objective Definition

```
constraint max_of_mins = max(client_min_distance);

solve minimize max_of_mins;
```

This constraint is responsible for choosing the maximum distance from all of the minimum possible distances between client/facility locations. Here is also described the fact that we solve the problem by optimizing the minimum of these values

1.3 Heuristics Usage

Some of the given data files will take too long to solve for the optimal solution. Therefore it's a valid idea to use heuristics to get a good enough solution in less time.

1.3.1 Variable Selection Heuristics

Variable selection heuristics decide which variable should be assigned a value next during the search process. Choosing the "right" variable early can significantly reduce the size of the search space and lead to faster solutions. In this assignment, the following heuristics were tested:

`First-Fail`: Selects the variable with the smallest domain first, aiming to fail early if a conflict arises.

`Anti-First-Fail`: Opposite of First-Fail; it selects the variable with the largest domain first to delay failures.

`Domain-Weighted Degree`: Chooses the variable with the highest ratio of domain size to the number of constraints it is involved in, giving preference to constrained but flexible variables.

1.3.2 Value Selection Heuristics

Value selection heuristics determine which value to try first when assigning a value to a variable. These heuristics help guide the search toward promising areas of the solution space. The goal is to quickly find consistent assignments and possibly prune inconsistent branches. The following value heuristics were applied:

`Indomain Min`: Always assigns the minimum value from the domain, potentially reaching a solution quickly when small values are preferred.

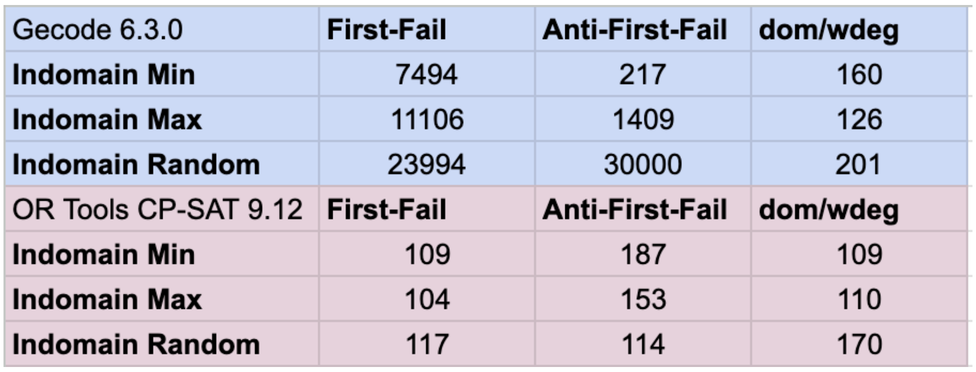
`Indomain Max`: Always selects the largest available value, which may be better when higher values are more likely to satisfy constraints.

`Indomain Random`: Picks a value at random from the domain, which can diversify the search and help escape patterns that cause failure.

1.4 Performance Results

To see how different heuristics affect performance, all combinations of variable and value selection heuristics were tested. Each combination was run on both the Gecode solver and OR Tools CP-SAT solver. The goal was to compare how fast each combination finds a solution and if it helps reduce solving time.

The following table shows the results for all combinations of variable and value heuristics on locs1.dzn file with a 30 second time limit (in ms):



Gecode 6.3.0	First-Fail	Anti-First-Fail	dom/wdeg
Indomain Min	7494	217	160
Indomain Max	11106	1409	126
Indomain Random	23994	30000	201
OR Tools CP-SAT 9.12	**First-Fail**	**Anti-First-Fail**	**dom/wdeg**
Indomain Min	109	187	109
Indomain Max	104	153	110
Indomain Random	117	114	170

As for the results, each of the heuristic combinations resulted in the same `max_of_mins` results, that being 5.

1.5 Execution Examples

1.5.1 Example 1

Input Data:

```
grid_size = 25;
num_clients = 3;
num_facilities = 2;
min_distance = [4,3];
distance_between_facilities = 3;

client_loc = [|4,15|12,4|23,15|];
```

Output:

```
 client_min_distance = [5, 4, 14];
 max_of_mins = 14;
----------
 client_min_distance = [6, 4, 13];
 max_of_mins = 13;
----------
 client_min_distance = [7, 4, 12];
 max_of_mins = 12;
----------
 client_min_distance = [8, 4, 11];
 max_of_mins = 11;
----------
 client_min_distance = [9, 4, 10];
 max_of_mins = 10;
----------
==========
```

1.5.2 Example 2

Input Data:

```
grid_size = 25;
num_clients = 5;
num_facilities = 2;
min_distance = [1,1];
distance_between_facilities = 4;

client_loc = [|1,5|2,4|3,5|1,3|4,5|];
```

Output:

```
 client_min_distance = [4, 2, 2, 2, 2];
 max_of_mins = 4;
----------
 client_min_distance = [2, 2, 2, 3, 2];
 max_of_mins = 3;
----------
==========
```

2 Exercise 2: Container Delivery

2.1 Problem Description

The container delivery problem involves scheduling trucks to load boxes from a port using derricks, transport them to warehouses, and unload them using cranes. The objective is to minimize the total completion time (makespan).

Key constraints:

- Each derrick takes 4 time units to load one box
- Limited number of derricks available at the port defined by `num_derricks`
- Travel time from port to each warehouse is fixed (wh1 = 15, wh2 = 20)
- Each warehouse has only one crane for unloading, which can unload one box per time unit

2.2 Solution Code

```
int: LOAD_TIME = 4;
int: UNLOAD_TIME = 1;

array[TRUCKS] of var TIME: StartLoad;
array[TRUCKS] of var TIME: StartUnload;
array[TRUCKS] of var TIME: End;

array[TRUCKS] of int: load_durations = [delivery[t, BOXES] * LOAD_TIME | t in
    TRUCKS];
array[TRUCKS] of int: unload_durations = [delivery[t, BOXES] * UNLOAD_TIME | t
    in TRUCKS];
```

The beginning section of the code defines decision variables for all of the time points which are important for each truck in their specific journey.

Here are also two constants for time required to load/unload boxes, and helper arrays for the total duration of load/unload times for each truck

Next up is the description of each constraint of the model.

1. Loading Constraint

```
constraint cumulative(StartLoad, load_durations, [1 | t in TRUCKS],
                      num_derricks);
```

This constraint models the limited capacity of derricks at each port. It uses the `cumulative` constraint, since there is a certain capacity of resources (derricks) that can be used at the same time.

`StartLoad` holds the output times at which each truck begins to load, `load_durations` gives the times required to load each truck, `[1 | t in TRUCKS]` says that each truck requires at least 1 derrick to be loaded, and `num_derricks` gives the capacity of derricks available in the port.

2. Unloading Constraint

```
constraint forall(w in WAREHOUSES) (
  let {
    set of int: WTRUCKS = {t | t in TRUCKS where delivery[t, DEST] == w}
  } in
  disjunctive([StartUnload[t] | t in WTRUCKS],
              [unload_durations[t] | t in WTRUCKS])
);
```

This constraint models the limited capacity of cranes at each warehouse. Each warehouse is modeled individually, and so the constraint starts by filtering which truck belongs to which warehouse. These filtered trucks are held in the `WTRUCKS` set.

It uses the `disjunctive` constraint, since there is no capacity of the cranes in each warehouse, just a single one running at each time unit, so the trucks have to stack up in case there are multiple in one warehouse.

`[StartUnload[t] | t in WTRUCKS]` holds the output times at which each truck begins to load and the `[unload_durations[t] | t in WTRUCKS]` gives the unload durations for each truck.

3. Arrival Time Constraint

```
constraint forall(t in TRUCKS) (
  StartUnload[t] >= StartLoad[t] + load_durations[t] +
                    travel_time[to_enum(WAREHOUSES, delivery[t, DEST])]
);
```

This constraint models the times at which each truck begins to unload. it doest it by counting up together the time at which trucks begins to loading, the duration of the load, and the travel time to the warehouse. The `StartUnload` time might be higher then that however, since the trucks might be stuck in queue while the crane is full.

4. Completion Time Constraint

```
constraint forall(t in TRUCKS) (
  End[t] = StartUnload[t] + unload_durations[t]
);
```

In the final constraint the unload duration is added to the `StartUnload` time from above.

5. Max time Constraint

```
var TIME: makespan;
constraint makespan = max([End[t] | t in TRUCKS]);
solve minimize makespan;
```

The final constraint gives the maximum time from the trucks to finish their job. This is also the constraint for which the model runs an optimization to find the minimal solution.

2.3 Execution Examples

2.3.1 Example 1

Input Data:

```
num_derricks = 3;
num_trucks = 4;
max_time = 100;

delivery = [|5,enum2int(wh1)
```

```
            |2,enum2int(wh2)
            |3,enum2int(wh1)
            |4,enum2int(wh2) |];
```

Output:

```
StartLoad = [0, 12, 0, 0]
StartUnload = [35, 40, 27, 36]
End = [40, 42, 30, 40]
_objective = 42
----------
==========
```

2.3.2 Example 2

Input Data:

```
num_derricks = 3;
num_trucks = 8;
max_time = 100;

delivery = [|5,enum2int(wh1)
            |2,enum2int(wh2)
            |3,enum2int(wh1)
            |4,enum2int(wh2)
            |5,enum2int(wh2)
            |6,enum2int(wh2)
            |6,enum2int(wh1)
            |7,enum2int(wh2)
          |];
```

Output:

```
StartLoad = [24, 44, 28, 40, 0, 0, 20, 0]
StartUnload = [65, 72, 55, 76, 40, 55, 59, 48]
End = [70, 74, 58, 80, 45, 61, 65, 55]
_objective = 80
----------
StartLoad = [24, 28, 44, 36, 0, 0, 20, 0]
StartUnload = [65, 61, 71, 72, 40, 55, 59, 48]
End = [70, 63, 74, 76, 45, 61, 65, 55]
_objective = 76
----------
StartLoad = [24, 44, 44, 28, 0, 0, 20, 0]
StartUnload = [65, 72, 71, 64, 40, 55, 59, 48]
End = [70, 74, 74, 68, 45, 61, 65, 55]
_objective = 74
----------
==========
```

2.3.3 Example 3

Input Data:

```
num_derricks = 4;
num_trucks = 10;
max_time = 200;

delivery = [|5,enum2int(wh1)
            |2,enum2int(wh2)
            |3,enum2int(wh1)
            |4,enum2int(wh2)
            |5,enum2int(wh2)
            |6,enum2int(wh2)
            |6,enum2int(wh1)
            |7,enum2int(wh2)
            |3,enum2int(wh2)
            |1,enum2int(wh2)
          |];
```

Output:

```
StartLoad = [28, 20, 28, 0, 0, 0, 24, 0, 16, 28]
StartUnload = [69, 64, 55, 36, 40, 55, 63, 48, 61, 66]
End = [74, 66, 58, 40, 45, 61, 69, 55, 64, 67]
_objective = 74
----------
StartLoad = [24, 20, 28, 0, 0, 0, 28, 0, 16, 28]
StartUnload = [59, 64, 55, 36, 40, 55, 67, 48, 61, 66]
End = [64, 66, 58, 40, 45, 61, 73, 55, 64, 67]
_objective = 73
----------
StartLoad = [28, 28, 24, 0, 0, 0, 20, 0, 16, 36]
StartUnload = [65, 64, 51, 36, 40, 55, 59, 48, 61, 66]
End = [70, 66, 54, 40, 45, 61, 65, 55, 64, 67]
_objective = 70
----------
StartLoad = [20, 28, 28, 0, 0, 0, 24, 0, 16, 36]
StartUnload = [58, 64, 55, 36, 40, 55, 63, 48, 61, 66]
End = [63, 66, 58, 40, 45, 61, 69, 55, 64, 67]
_objective = 69
----------
StartLoad = [24, 32, 20, 0, 0, 0, 16, 0, 28, 40]
StartUnload = [61, 64, 47, 36, 40, 55, 55, 48, 61, 66]
End = [66, 66, 50, 40, 45, 61, 61, 55, 64, 67]
_objective = 67
----------
StartLoad = [24, 36, 24, 0, 0, 0, 16, 0, 28, 20]
StartUnload = [61, 64, 51, 36, 40, 55, 55, 48, 61, 45]
End = [66, 66, 54, 40, 45, 61, 61, 55, 64, 46]
_objective = 66
----------
==========
```